#include <iostream>

#include <string>

using namespace std;

class Mama{

public:

Mama(){id = -1;}

Mama(const int & anId){id = anId;}

Mama(const Mama & anotherMama){id = anotherMama.id;}

void setId(const int & anId){id = anId;}

virtual void show(){cout<<"\t\tMama "<< id <<endl;}

protected:

int id;

};

class Child: public Mama{

public:

Child():Mama(){childId = -2;}

Child(const int & anId, const int & aCId):Mama(anId){childId = aCId;}

Child(const Child & anotherChild){id = anotherChild.id; childId = anotherChild.childId;}

void show(){cout << "\t\tChild " << id << " " << childId <<endl;}

protected:

int childId;

};

class MamaManager{

public:

void setArrayMAtPos(int pos, const Mama & aMama){arrayMama[pos] = const\_cast<Mama &>(aMama);}

void setArrayCAtPos(int pos, const Child & aChild){arrayChild[pos]= const\_cast<Child &>(aChild);}

void setArrayPMAtPos(int pos, const Mama \* anAddress){arrayPMama[pos]= const\_cast<Mama \*>(anAddress);}

void testAll();

void detailMama(Mama m); **//intentionally, no &, no const**

void detailMamaP(Mama \* m);

private:

Mama arrayMama[2];

Child arrayChild[2];

Mama \* arrayPMama[2];

};

void MamaManager::detailMama(Mama m){

cout << "Details: "; m.show();

}

void MamaManager::detailMamaP(Mama \* m){

cout << "Details: "; m->show();

}

void MamaManager::testAll(){

cout << "First the array of Mamas" << endl;

for(int i =0; i < 2; i++)

arrayMama[i].show();

cout << "\n\nThen, the array of Child" << endl;

for(int i =0; i < 2; i++)

arrayChild[i].show();

cout << "\n\nThen, the array of pointers" << endl;

for(int i =0; i < 2; i++)

arrayPMama[i]->show();

}

int main(){

Mama m(100); Child c(150, 200); MamaManager g;

g.setArrayMAtPos(1, m);

g.setArrayCAtPos(1,c);

g.setArrayPMAtPos(0, &m);

g.setArrayPMAtPos(1, &c);

cout << "\nDetail a Mama" << endl;

g.detailMama(m);

cout << "\nDetail a Child" << endl;

g.detailMama(c);

cout << "\nDetail a MamaPointer" << endl;

Mama \* mP = &m;

g.detailMamaP(mP);

cout << "\nDetail a ChildPointer" << endl;

Child \* cP = &c;

g.detailMamaP(cP);

cout << "\nDetail a Mama2ChildPointer " << endl;

mP = &c;

g.detailMamaP(mP);

cout << "\n\nNow, go through all the members of the manager\n";

g.testAll();

return 0;

}

Detail a Mama

Details: Mama 100

Detail a Child

Details: Mama 150

Detail a MamaPointer

Details: Mama 100

Detail a ChildPointer

Details: Child 150 200

Detail a Mama2ChildPointer

Details: Child 150 200

Now, go through all the members of the manager

First the array of Mamas

Mama -1

Mama 100

Then, the array of Child

Child -1 -2

Child 150 200

Then, the array of pointers

Mama 100

Child 150 200

|  |  |
| --- | --- |
| #include <iostream>  #include <string>  using namespace std;  class Mama{  public:  **virtual** void show() **=0**;  };  class Child1: public Mama{  public:  void show(){cout<<"Child1" <<endl;}  };  class Child2: public Mama{  public:  void show(){cout<<"Child2" <<endl;}  };  int main(){  int choice = 100; **Mama \* ptr** = NULL;  do{  cout << "Give 0 (mama), 1(child1), 2(child2), -1(to exit): ";  cin >> choice;  if (choice == -1)  break;  **//cannot do the following if show is virtual => Mama is an abstract class**  **// if (choice==0){ ptr = new Mama; ptr->show();}**  if (choice==1){  ptr = new Child1; ptr->show();  }  else if (choice==2){  ptr = new Child2; ptr->show();  }  else cout << "Wrong choice!" << endl;  }while(choice !=-1);  return 0;  } |  |
| $ ./simple  Give 0 (mama), 1(child1), 2(child2), -1(to exit): 0  Wrong choice!  Give 0 (mama), 1(child1), 2(child2), -1(to exit): 1  Child1  Give 0 (mama), 1(child1), 2(child2), -1(to exit): 2  Child2  Give 0 (mama), 1(child1), 2(child2), -1(to exit): -1 |  |

#include <iostream>

#include <string>

#include <set>

using namespace std;

// \*\*\*\*\*\*\*\*\*\* DEVELOPER #1:

//Builds a hierarchy of items. item is abstract, is giving only a 'contract' of what items do

//so that all subclasses implement the 'contract' and any client program, like the ItemMgr next

//does NOT need to know the details of the subclasses, but only the 'contract' of the mama abstract class

class Item{

public:

virtual void setPrice(const int & aPrice) = 0;

virtual void show() =0;

};

class Book: public Item{

public:

void setPrice(const int & aPrice){price = 1.23\*aPrice;}

void show(){cout<<"Book "<<price <<endl;}

private:

double price;

};

class CD: public Item{

public:

void setPrice(const int & aPrice){price = aPrice;}

void show(){cout<<"CD " <<price <<endl;}

private:

int price;

};

// \*\*\*\*\*\*\*\*\*\* DEVELOPER #2:

//he makes the item manager, that uses the hierarchy of Items and hides everything from the client.

//Still: he needs help from a specific class that knows all the trouble with the hierarchy of Items

//so that the manager is independent from the complexities of the hierarchy

//Therefore: we have a 'factory' class, which is the ONLY point where the hierarchy module and the

//mgr module meet. factory depends on ALL subclasses of Item, needs maintenance every time that a new

//subclass is added; still, this is the ONLY place in the code of dvlpr #2 that this happens.

//This way, the mgr is independent from the hierarchy.

class ItemFactory{

public:

Item \* addItem();

};

Item \* ItemFactory::addItem(){

int choice, price; Item \* newItem;

cout << "\nGive 1 for Book and 2 for CD" << endl;

cin >> choice;

cout << "Give a price: ";

cin >> price;

if (choice==1){

newItem = new Book; newItem->setPrice(price);

}

else if (choice==2){

newItem = new CD; newItem->setPrice(price);

}

else newItem = NULL;

return newItem;

}

class ItemMgr{

public:

void addItem();

void listAll();

private:

ItemFactory iFactory;

set<Item \*> items;

};

void ItemMgr::addItem(){

Item \* iPtr;

iPtr = iFactory.addItem();

if (iPtr != NULL) items.insert(iPtr);

}

void ItemMgr::listAll(){

set<Item \*>::iterator iIt;

for (iIt=items.begin(); iIt != items.end(); iIt++){

(\*iIt)->show();

}

}

// \*\*\*\*\*\*\*\*\*\*\* DEVELOPER #3

//and an actual client, knows nothing really for the hierarchy and lets the boss do everything.

//btw: for this, we will need to encompass in the mgr, some methods that do tasks that are

//delegated to the mgr's items inside the mgr. Here: to do the printing, mgr exports 'listAll()'

//that internally invokes all items' show()

int main(){

ItemMgr theBoss;

for (int i=0;i<3;i++)

theBoss.addItem();

cout <<"\n\n ... and so, we have loaded the following items:" << endl;

theBoss.listAll();

return 0;

}

|  |
| --- |
| $ ./factory.exe  Give 1 for Book and 2 for CD  1  Give a price: 10  Give 1 for Book and 2 for CD  1  Give a price: 20  Give 1 for Book and 2 for CD  2  Give a price: 45  ... and so, we have loaded the following items:  Book 12.3  Book 24.6  CD 45 |